DİZİLER VE LİSTELER

Dr. Öğr. Üyesi Nilgün Şengöz

ALGORİTMALAR

- Python'da listeler ve diziler birden fazla öğeyi depolamak için kullanılan veri yapılarıdır.
- Her ikisi de, öğelere erişmek için öğelerin indekslenmesini, dilimlenmesini ve öğeler üzerinde yineleme yapılmasını destekler.
- Özellikle dizi ve listenin ne zaman kullanılacağını nasıl anlarız?
- Bu durum için dizilerin ve listelerin ne olduğunu detaylı bir şekilde bilmemiz gerekmektedir.

DİZİLER (ARRAYS)

- Diziler, Python ve diğer birçok programlama dilinde sıkça kullanılan veri yapılarıdır
- Diziler, aynı türdeki verilerin bir koleksiyonunu saklar
- Diziler, homojen öğeler içeren, yani aynı veri türüne ait bir vektördür.
- Öğeler bitişik bellek konumlarıyla tahsis edilir. Tipik olarak bir dizinin boyutu sabittir.
- Ekleme ve silme maliyetleri listeye göre yüksektir ancak bitişik bellek tahsisi nedeniyle dizilerde indeksleme daha hızlıdır.
- Diziler, dizi modülü içe aktarılarak kullanılabilir.

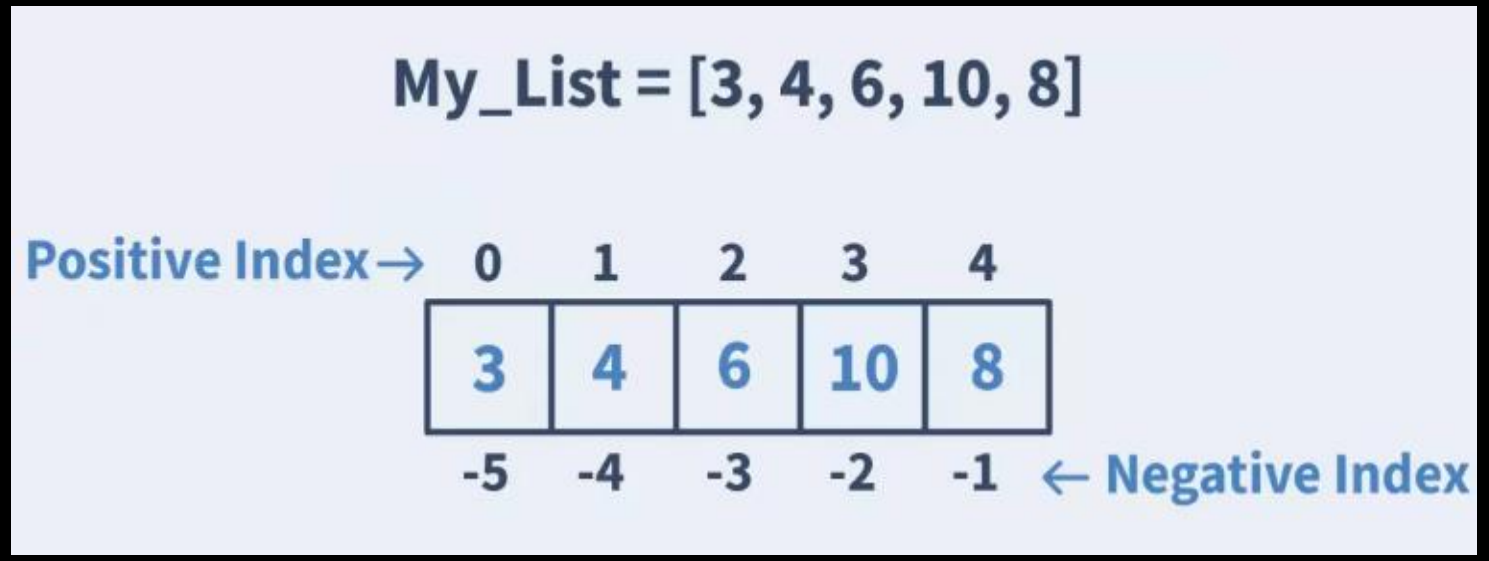
LİSTELER (LISTS)

- Listeler (lists) temel veri yapılarından biridir.
- Listeler, birden çok öğeyi içinde depolayabilen ve değiştirilebilir (mutable) bir veri tipidir.
- Esnekliği ve desteklediği zengin işlemler kümesi nedeniyle Python'un en çok yönlü ve yaygın olarak kullanılan veri yapılarından biridir.
- Listeler, ekleme sırasına göre öğelerin sırasını korur.
- Listeler sayılar, dizeler ve diğer nesneler dahil olmak üzere farklı veri türlerindeki öğeleri depolayabilir.

LİSTELER (LISTS)

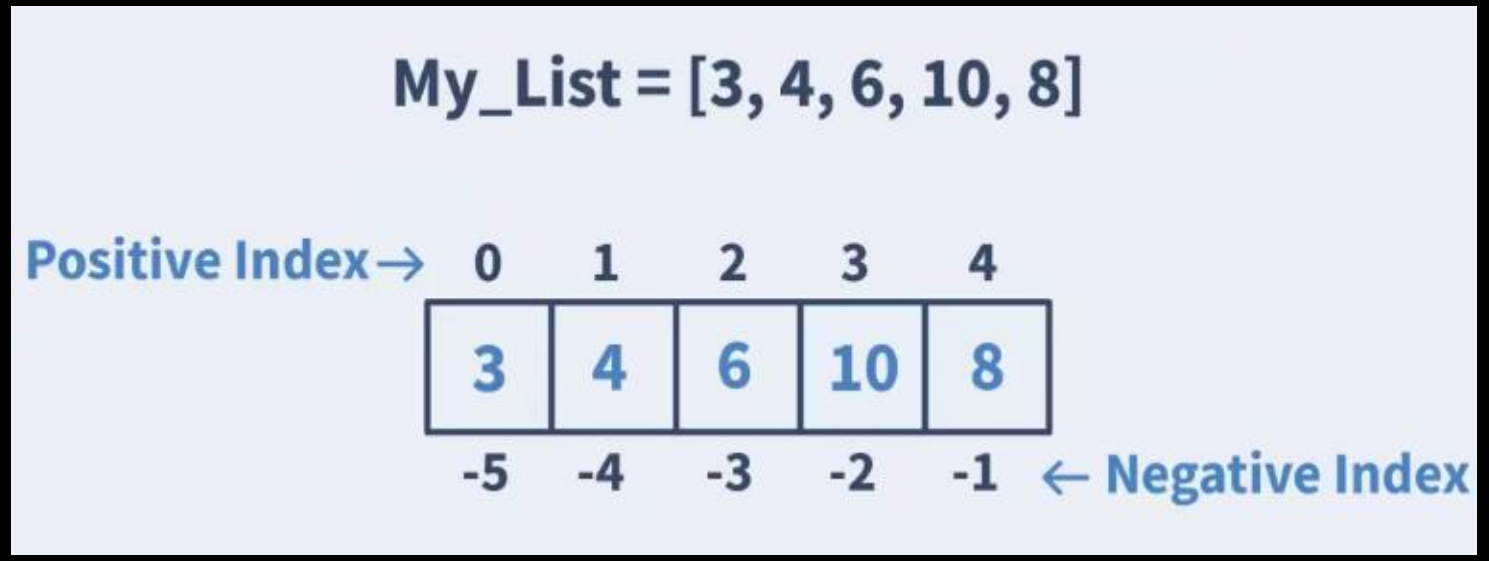
- Listeler değiştirilebilir; yani içeriklerini oluşturulduktan sonra değiştirebilirsiniz. Bir listeye öğe ekleyebilir, kaldırabilir veya değiştirebilirsiniz.
- Listelerin boyutu dinamik olarak büyüyebilir veya küçülebilir. Öğeleri istediğiniz konuma ekleyebilir ve gerektiği gibi öğeleri kaldırabilirsiniz.
- Listeler indekslemeyi (tek tek öğelere erişmek için) ve dilimlemeyi (bir dizi öğeye erişmek için) destekler. Dizin oluşturma ilk öğe için 0'dan başlar.
- Python, listeler için Append(), Remove(), Sort(), Reverse() ve çok daha fazlası gibi çok sayıda yerleşik yöntem sağlar.
- Listeler başka listeler içerebilir, bu da iç içe geçmiş veya çok boyutlu listelerle sonuçlanır.
- Listelerde yinelenen öğeler bulunabilir. Aynı değeri birden çok kez saklama konusunda herhangi bir kısıtlama yoktur.

LİSTELER (LISTS)



- Listelerde saklanan öğeler, indeks olarak bilinen benzersiz bir tamsayı numarasıyla ilişkilendirilir. İlk öğe 0, ikincisi 1 vb. olarak indekslenir. Yani, altı öğe içeren bir listenin 0'dan 5'e kadar bir dizini olacaktır. Listedeki öğelere erişim için, dizin, listenin adından önce dizin operatörü ([ ]) içinde belirtilir.

LİSTELER (LISTS)



- Python'daki listedeki öğelere/değerlere erişmenin başka bir yolu da negatif dizin kullanmaktır.
- Negatif indeks son öğe için -1'den, son ikinci öğe için -2'den başlar ve bu şekilde devam eder.

LİSTELER (LISTS)

```
My_List = [3, 4, 6, 10, 8]

# Listedeki öğelere pozitif indeksleme ile erişme
print("Pozitif indeks değerlerine erişildi.")
print(My_List[2])
print(My_List[4])

# Listedeki öğelere negatif indeksleme ile erişme
print("Negatif indeks değerlerine erişildi.")
print(My_List[-1])
print(My_List[-5])
```

```
Pozitif indeks değerlerine erişildi.
6
8
Negatif indeks değerlerine erişildi.
8
3
```

- Listedeki elemanların öğelerine sağdan ve soldan yaklaşım yani pozitif ve negatif indeksleme yoluyla erişim sağlama

LİSTELER (LISTS)

```
# İç içe listeler
nested_List = [[6, 2, 8], [1, 3.5, "Röportaj"], "hazırlık"]
print(nested_List[0][1])
print(nested_List[1][2])
```

```
2
Röportaj
```

- İç içe Listeler için, öncelikle indeks operatörünü kullanarak iç listenin dış listede saklandığı/indekslendiği indekse erişiriz. Daha sonra tekrar indeks operatörünü kullanarak iç listeden elemana ulaşıyoruz.
- Bu duruma «iç içe» indeksleme denilir. Sanki 2 boyutlu bir dizideki öğelere erişiyoruz. Bundan farklı olarak, iç listede birçok öğemiz var, ancak dizilerin o indekste yalnızca tek bir değeri var.

LİSTELER (LISTS)

```
My_List = [3, 4, 6, 10, 8, 5]
print(My_List)
# indeks 3'teki değer değiştiriliyor.
My_List[3] = 7
print(My_List)
# -1 indeksinde yer alan değer değiştiriliyor
My_List[-1] = 15
print(My_List)
```

```
[3, 4, 6, 10, 8, 5]
[3, 4, 6, 7, 8, 5]
[3, 4, 6, 7, 8, 15]
```

- Listeler değiştirilebilir, yani indekslerde saklanan mevcut değerleri değiştirilebilir. Liste öğelerini değiştirmek veya güncellemek için atama operatörü (=) kullanılır.

LİSTELER (LISTS)

```
vowels = ['a', 'e']
print(vowels)
# Listeye eleman ekleme
vowels.append('i')
print(vowels)
# listeyi genişletme
vowels.extend(['o', 'u'])
print(vowels)
# Belirli bir indekse eleman sokmak
Integers = [1, 5, 6, 9]
Integers.insert(1,7)
print(Integers)
```

```
['a', 'e']
['a', 'e', 'i']
['a', 'e', 'i', 'o', 'u']
[1, 7, 5, 6, 9]
```

- Listeler dizelerden farklı olarak değiştirilebilir olduğundan listelere öğeler ekleyebiliriz. Listenin sonuna tek bir öğe eklemek için Append() yöntemini, listenin sonuna birden fazla öğe eklemek için Extend() yöntemini kullanırız.
- Ayrıca belirtilen dizin konumuna bir öğe eklemek için bir insert() yöntemimiz de var. Insert() yöntemi iki argüman alır; birincisi öğenin ekleneceği dizin, ikincisi ise eklenecek değer/öğedir.

LİSTELER (LISTS)

```
my_List = [1, 2, 3, 4, 5, 6, 7, 8]
print(my_List)
# Listedeki elemanın silinmesi
del my_List[3]
print(my_List)
# Tüm listenin silinmesi
del my_List
# print() kodu bize hata mesajı verecek
print(my_List)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 5, 6, 7, 8]

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 9
      7 del my_List
      8 # print() kodu bize hata mesajı verecek
----> 9 print(my_List)

NameError: name 'my_List' is not defined
```

- Python'da del anahtar sözcüğünü kullanarak listeden bir veya daha fazla öğeyi silebiliriz. Hatta bu anahtar kelimeyi kullanarak tüm listeyi silebiliriz. Bir listeyi silersek ve sonra ona erişmeye çalışırsak yorumlayıcı bir hata üretecektir.

```python
my_List = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(my_List)
# remove metodu kullanarak öğrenin silinmesi
my_List.remove(4)
print(my_List)
# pop metodu kullanarak öğrenin silinmesi
print(my_List.pop(1))
print(my_List)
# herhangi bir indeks belirtilmeden pop() methodu kullanımı
print(my_List.pop())
print(my_List)
# clear() methodu kullanımı
my_List.clear()
print(my_List)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 5, 6, 7, 8, 9]
2
[1, 3, 5, 6, 7, 8, 9]
9
[1, 3, 5, 6, 7, 8]
[]
```

- Remove() yöntemi, öğeyi silinmesi veya kaldırılması gereken bir argüman olarak alır. Öğe listede yoksa ValueError hatası ortaya çıkar.
- pop() yöntemi, dizini/indeksi argüman olarak alır ve silinen öğeyi siler ve döndürür. Eğer indeks belirtilmemişse, pop() listenin son elemanını siler ve döndürür.
- Python'daki clear() yöntemi listeyi boş yapar.

```
List1 = [1, 2, 3, 4]
List2 = [5, 6, 7, 8]
concat_List = List1 + List2
print(concat_List)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

- Birleştirme: Bir liste kendisiyle veya başka bir listeyle '+' operatörü kullanılarak birleştirilebilir.

```
my_List = [1, 2, 3, 4]
print(my_List*2)
```

```
[1, 2, 3, 4, 1, 2, 3, 4]
```

- Listenin elemanlarını tekrarlamak: Liste elemanlarını istediğimiz kadar tekrarlamak için '*' operatörünü kullanabiliriz.

```
my_List = [1, 2, 3, 4, 5, 6]
print(3 in my_List)
print(10 in my_List)
print(10 not in my_List)
```

```
True
False
True
```

- Listedeki bir elemanın Üyelik Kontrolü: Belirli bir elemanın listenin parçası/üyesi olup olmadığını kontrol edebiliriz.

```
my_List = [1, 2, 3, 4, 5]
for n in my_List:
        print(n)
```

```
1
2
3
4
5
```

- Liste boyunca yineleme: Bir for döngüsü kullanarak listedeki her öğeyi yineleyebiliriz.

```
my_List = [1, 2, 3, 4, 5]
print(len(my_List))
```

```
5
```

- Python'da Listenin uzunluğunu bulma: Listenin uzunluğunu bulmak için len() yöntemi kullanılır.

```python
my_List = ['i', 'n', 't', 'v', 'i', 'e', 'w', 'b', 'i', 't']
# 6. indekse kadarm olan öğeler
print(my_List[:6])
# 3-6 indeks arası öğeler
print(my_List[3:7])
# 5. indeksten listenin sonuna kadar olan öğeler
print(my_List[5:])
# baştan sona olan tüm öğeler
print(my_List[:])
# negatif yönde olan listeleme
print(my_List[:-6])
```

```
['i', 'n', 't', 'v', 'i', 'e']
['v', 'i', 'e', 'w']
['e', 'w', 'b', 'i', 't']
['i', 'n', 't', 'v', 'i', 'e', 'w', 'b', 'i', 't']
['i', 'n', 't', 'v']
```

- Python'da dilimleme operatörünü (:) kullanarak bir dizi indekste saklanan öğelere erişebiliriz. Dilimleme operatörünün sol tarafına yerleştirilen indeks dahil olup, sağ tarafta belirtilenler hariçtir.

- Dilimin kendisi bir şeyin parçası anlamına gelir. Dolayısıyla bir listenin belirli bir bölümünde veya aralığında saklanan öğelere erişmek istediğimizde dilimlemeyi kullanırız.

```
List = [1, 2, 3, 4, 5, 6, 7, 8]
print(List)
# 2. indeksten 5. indekse karad olan elemanların değişimi
List[2:5] = [1, 1, 1]
print(List)
# listeden elemanları silme
del List[2:6]
print(List)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 1, 1, 1, 6, 7, 8]
[1, 2, 7, 8]
```

- Listedeki birden fazla öğeyi değiştirmek, kaldırmak/silmek için dilimlemeyi de kullanabiliriz. Bazı kod örneklerini görelim ve anlayalım.

Özellik	Diziler	Listeler
Homojenlik	Tüm öğeler aynı veri tipinde olmalı	Farklı veri tipleri içerebilir (heterojendir)
Boyut Esnekliği	Bir kez oluşturulduktan sonra sabit boyutlu.	Dinamik boyut; ihtiyaca göre büyüyebilir veya küçülebilir.
Bellek Verimliliği	Homojenlik nedeniyle daha verimli bellek.	Potansiyel tür değişkenliği nedeniyle bellek açısından daha az verimli.
Yerleşik Yöntemler (Built-in Methods)	Daha az yerleşik yöntem.	Çeşitli işlemler için zengin yerleşik yöntemler kümesi.
Modül Gereksinimi	Dizi modülünün içe aktarılmasını gerektirir.	Yerleşik veri yapısı; import edilmesine gerek yok.
Tip Spesifikasyonu	Oluşturma sırasında bir tür kodu gerektirir (örneğin, int için 'i').	Herhangi bir tip spesifikasyonuna gerek yoktur.
Kullanım	Sayısal işlemler için ve C kütüphaneleriyle arayüz oluştururken uygundur.	Genel amaç; çok çeşitli görevler için uygundur.
Veri Erişimi	Dizin kullanarak doğrudan erişim.	Dizin kullanarak doğrudan erişim.
İç içe yerleştirme/Gömme	Yalnızca belirtilen türdeki öğeleri içerebilir.	Diğer listeler (iç içe) dahil olmak üzere herhangi bir türü içerebilir.
Çoğaltma	Yinelenen öğelere sahip olabilir.	Yinelenen öğelere sahip olabilir.
Gösterimi	Dizi modülünü kullanır.	Köşeli parantez [ ] kullanılarak temsil edilir.
Verim/Performans	Bitişik bellek depolaması nedeniyle belirli senaryolarda daha iyi performans sunabilir.	Çok yönlüdür ancak tür denetimi nedeniyle belirli işlemlerde dizilerden daha az performanslı olabilir.